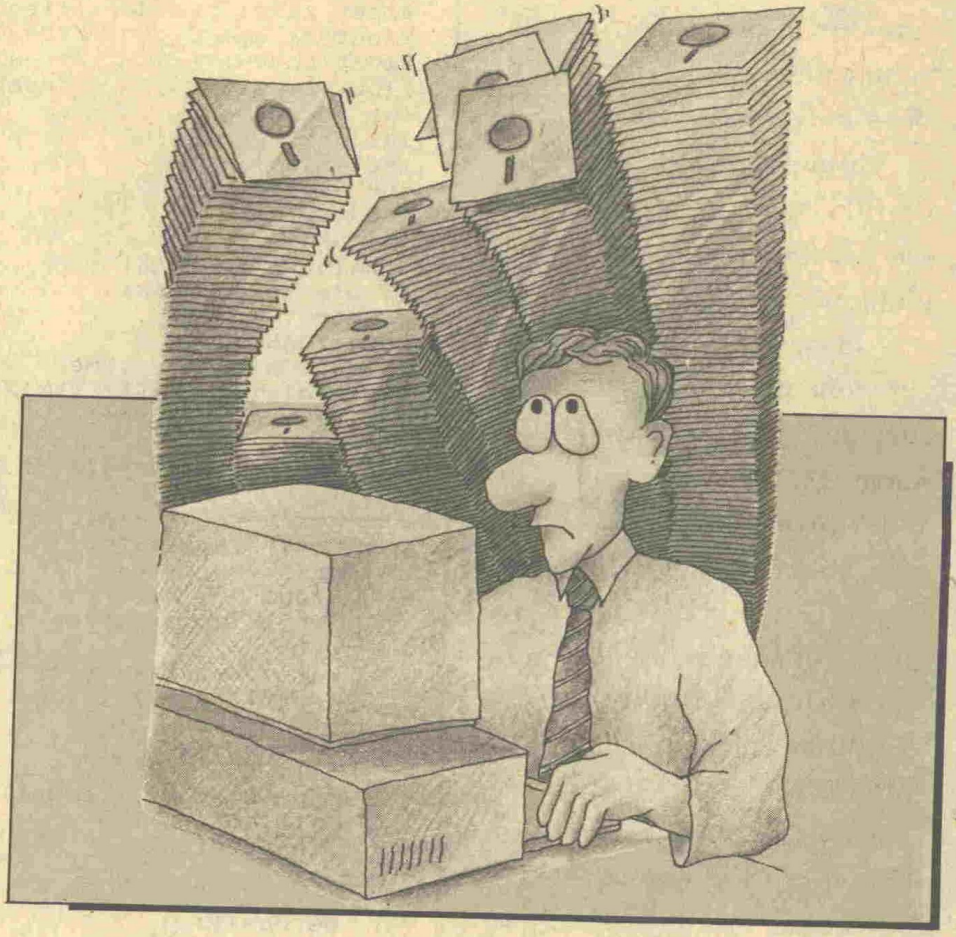
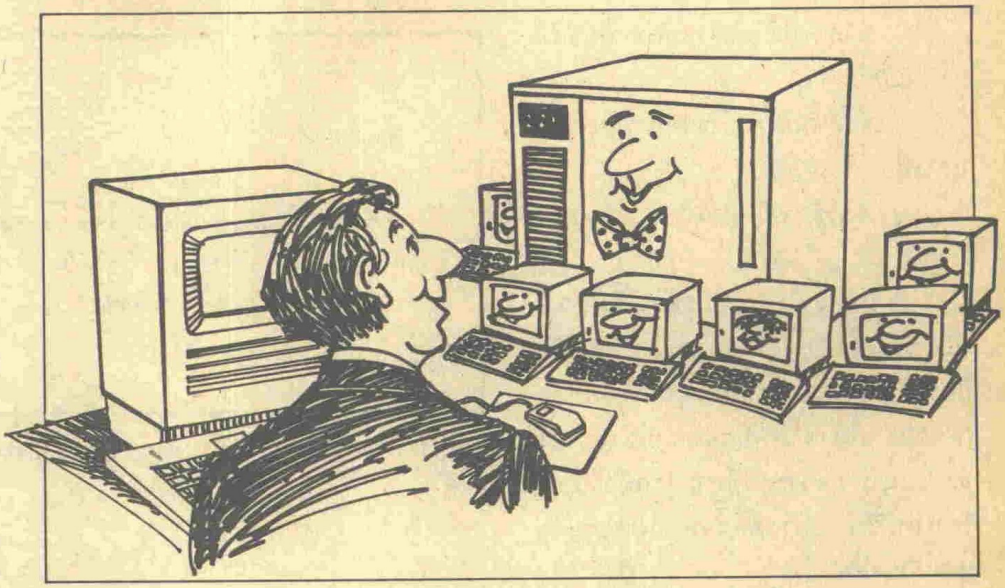
ซอฟต์แวร์
วิสันต์ เลาหสุขไพศาล

การใช้งานหน่วยความจำแบบ Extended ด้วยภาษาซี

เป็นเทคนิคการใช้งานหน่วยความจำที่ติดตั้งเอาไว้เหนือแอดเดรส



Extended Memory หมายถึงหน่วยความจำที่ติดตั้งไว้อยู่เหนือแอดเดรส 1 เมกะไบต์ ของซีพียูตระกูล 80x86 ซึ่งจะเป็นซีพียู 80286, 80386, 80486 ส่วนซีพียู 8086/8088 ไม่สามารถที่จะมีหน่วยความจำส่วนนี้ได้เนื่องจากว่า 8086/8088 สามารถที่จะอ้างอิงหน่วยความจำได้สูงสุดเพียง 1 เมกะไบต์ ซึ่งเครื่องที่มีขายในท้องตลาดในปัจจุบันแทบจะทั้งหมดจะติดตั้งซีพียู 80286, 80386, 80486 และติดตั้งหน่วยความจำมาให้ตั้งแต่ 1, 2, 4, 8 เมกะไบต์แล้วแต่ความต้องการของผู้ซื้อ และเครื่องพวกนี้โดยมากมักจะใช้ IBM-DOS/MS-DOS เป็นระบบปฏิบัติการ อาจจะมีบ้างที่ใช้ OS/2, WINDOW NT, ยูนิกซ์ และการที่ใช้ดอสนี่เองซึ่งมีข้อจำกัดเกี่ยวกับการใช้หน่วยความจำคือว่าดอสถูกจำกัดอยู่ภายใต้หน่วยความจำเพียง 640 กิโลไบต์ ซึ่งอยู่ระหว่างแอดเดรส 0-640 กิโลไบต์ ส่วนที่อยู่เหนือ 640 กิโลไบต์ถึง 1024 กิโลไบต์จะถูกจับจองโดย Display Memory และรอมไบออส ดังนั้นแม้เครื่องคอมพิวเตอร์จะติดตั้งหน่วยความจำไว้มากก็ตาม โปรแกรมที่คุณพัฒนาขึ้นมาภายใต้ดอสก็ไม่สามารถใช้หน่วยความจำเกิน 640 กิโลไบต์ได้ ดังนั้นจึงได้เกิดการพัฒนาเทคนิคการใช้หน่วยความจำขึ้นเช่น EMS (Expanded Memory Specification) ซึ่งเป็นการเพิ่มหน่วยความจำให้กับโปรแกรมได้เช่นเดียวกัน สำหรับเวอร์ชันของ EMS ที่ได้มีการใช้แพร่หลายคือ LIM EMS 4.0 ซึ่งเริ่มมีการนำมาใช้เมื่อประมาณ 5 ปีที่แล้ว แต่การที่จะใช้งาน EMS นี้ก็จำเป็นที่จะต้องมีเทคนิคการเขียนเพื่อการใช้งานโดยเฉพาะแต่ผมจะไม่กล่าวถึงเนื่องจากคงจะมีผู้ที่เคยกล่าวถึงไว้แล้วในตอนก่อน ๆ แต่ผมจะได้กล่าวถึงการใช้งานหน่วยความจำแบบ Extended ภายใต้ XMS ซึ่งจัดเป็นเทคนิคการใช้งานหน่วยความจำแบบหนึ่ง ต่อไปเรามาทำความรู้จักกับ XMS กันดีกว่า XMS คืออะไร



XMS คืออะไร

Extended Memory Specification (XMS) เป็นข้อกำหนดทางด้าน Software Interface สำหรับการใช้งานหน่วยความจำที่ดอสไม่สามารถจัดการได้ คือว่าฟังก์ชัน DOS ไม่สามารถที่จะขอจองมาใช้งานได้ แล้วหน่วยความจำส่วนใดบ้างที่ดอสไม่สามารถจัดการได้

Upper Memory Blocks (UMB)

Upper Memory Blocks (UMB) คือหน่วยความจำที่อยู่ระหว่างแอดเดรส 640 กิโลไบต์ถึง 1024 กิโลไบต์ ซึ่งหน่วยความจำส่วนนี้จะสามารถนำมาใช้งานได้ภายใต้การเรียกใช้ผ่าน XMS Driver แต่หน่วยความจำส่วนนี้มักจะมีน้อย เนื่องจากถูกจับจองโดยรอมไบออสและ DISPLAY MEMORY ไปจนเกือบหมด

High Memory Area (HMA)

High Memory Area (HMA) คือหน่วยความจำที่อยู่ระหว่างแอดเดรส 1024 กิโลไบต์ถึง 1088 กิโลไบต์ มีขนาด 64 กิโลไบต์ หน่วยความจำส่วนนี้จะแปลกกว่าเพื่อนคือว่า มันสามารถที่จะถูกนำมาใช้งานโดยซีพียูใน Real Mode ได้ ซึ่งโดยปกติแล้วการที่จะใช้งานหน่วยความจำที่มีแอดเดรสเกิน 1 เมกะไบต์จะต้องทำการเปลี่ยนโหมดการทำงานของซีพียูเป็นแบบ Protected Mode ก่อนจึงจะสามารถใช้งานหน่วยความจำส่วนนั้นได้ ซึ่งเทคนิคนี้คือการใช้งาน A20 address ซึ่งโดยปกติแล้วการทำงานภายใต้ Real Mode จะใช้แอดเดรสตั้งแต่ A0-A19 ส่วน A20 จะมีเฉพาะซีพียูตั้งแต่ 80286 ขึ้นไป แต่ใน Real Mode ส่วนนี้จะไม่ทำงาน และเนื่องจากการอ้างอิงหน่วยความจำของซีพียู 80x86 เป็นแบบเซกเมนต์และ Offset คือจะสามารถหา Physical Address โดยการคูณค่าเซกเมนต์ด้วย 16 แล้วบวกด้วยค่า Offset ซึ่งค่าที่ได้นี้หากเกิน 20 บิต ก็จะถูกตัดทิ้งไป นั่นคือแอดเดรสจะอยู่ระหว่างค่า 00000H-FFFFFH ดังนั้นถ้าแอดเดรสเป็น 0-FFFF:0010H ซึ่งถ้าทำการหา Physical address จะเท่ากับ 000000H ถ้าแอดเดรส A20 ไม่ถูกใช้งาน แต่ถ้า A20 ถูกใช้งานจะหมายถึงแอดเดรส 100000H และหน่วยความจำพิเศษจำนวน 65,520 ไบต์ที่เรียกว่า HMA ก็จะมาจาก Physical address 100000-10FFEFH ซึ่งถ้า A20 ทำงานการอ้างอิงถึง Physical address 00000-0FFEFH จะถูกแม็พไปที่แอดเดรส 100000-10FFEFH แทน ซึ่งการควบคุมการทำงานของ A20 จะสามารถควบคุมได้โดยผ่าน XMS Driver โดยปกติเราก็สามารถขอใช้หน่วยความจำส่วนนี้มาใช้ได้ แต่เนื่องจากดอส 5.0 สามารถที่จะใช้งานหน่วยความจำส่วนนี้ได้อย่างมีประสิทธิภาพอยู่แล้วเราจึงไม่ควรนำมาใช้อีก

Extended Memory Block (EMB)

Extended Memory Blocks (EMB) คือหน่วยความจำที่อยู่เหนือแอดเดรส 1088 กิโลไบต์ ซึ่งจะมีประโยชน์ในการนำมาใช้งานได้โดยการใช้งานผ่าน XMS Driver โดยการเรียกผ่านฟังก์ชัน call ได้

ฟังก์ชันของ XMS Driver

แต่ก่อนที่จะสามารถใช้งาน XMS ได้เราจำเป็นต้องติดตั้ง XMS Driver ก่อน ซึ่งในดอส 5.0 ได้มีไว้ให้อยู่แล้ว คือ HIMEM.SYS ซึ่ง HIMEM.SYS จะมีหน้าที่ดูแลหน่วยความจำแบบ Extended ทั้งหมดที่ติดตั้งไว้ รวมทั้งจะเป็นตัวให้ฟังก์ชันสำหรับการใช้งานด้วย คราวนี้เรามาดูว่า XMS Driver มีฟังก์ชันอะไรให้บริการบ้างสำหรับ XMS Driver นี้จะสามารถจัดแบ่งได้เป็น 5 กลุ่มคือ

- ฟังก์ชันใช้ในการดูข้อมูลของไดรเวอร์
- ฟังก์ชันใช้ในการจัดการเกี่ยวกับ HMA

- ฟังก์ชันใช้ในการจัดการเกี่ยวกับขา A20
- ฟังก์ชันใช้ในการจัดการเกี่ยวกับ EMB
- ฟังก์ชันใช้ในการจัดการเกี่ยวกับ UMB

สำหรับฟังก์ชันที่เขียนขึ้นมานี้เป็นฟังก์ชันการใช้งานหน่วยความจำแบบ Extended ที่เป็นส่วนที่เรียกว่า (EMB) เนื่องจากหน่วยความจำส่วนนี้จะเป็นหน่วยความจำที่สามารถมีจำนวนได้มาก ขึ้นกับการติดตั้งหน่วยความจำสำหรับเครื่องนั้น ๆ

รายชื่อไฟล์

- XMS.H	header file
- XMSROU.C	source for xms function
- XMSL.LIB	library for xms function large model (TC20)
- XMSTEST.C	test program

รูปที่ 1 ตัวอย่างไลบรารีที่เป็น Large model library

หน้าที่ของแต่ละฟังก์ชัน

XmsAccess( ) ฟังก์ชันนี้ใช้ในการตรวจสอบว่าได้มีการติดตั้ง XMS Driver ไว้หรือไม่

XmsInfo( ) ฟังก์ชันนี้ใช้ในการดูว่า XMS เวอร์ชันเท่าไหร่และ XMS Driver revision เท่าไหร่

XmsMemoryAvailable( ) ฟังก์ชันนี้ใช้ในการตรวจสอบว่ามีหน่วยความจำ EMB เหลืออยู่เท่าไหร่และขนาดที่ใหญ่ที่สุดที่สามารถขอจองมาใช้ได้

XmsAllocate( ) ฟังก์ชันนี้ใช้ในการขอจองหน่วยความจำ EMB มาใช้งาน ค่าที่รีเทอร์นกลับมาจะเป็นค่าจำนวนเต็มค่าหนึ่ง แต่ค่านี้ต้องไม่เท่ากับ 0 ซึ่งค่านี้จะนำมาใช้สำหรับฟังก์ชันสำหรับการย้ายข้อมูล เมื่อใช้งานหน่วยความจำส่วนนี้แล้วควรจะต้องใช้ XmsRelease( ) ทุกครั้ง

XmsReAllocate( ) ฟังก์ชันนี้ใช้ในการขยายหน่วยความจำที่เราเคยจองไว้ก่อนโดยฟังก์ชัน XmsAllocate( )

XmsLockhandle( ) ฟังก์ชันนี้ใช้ในการล็อกหน่วยความจำ EMB ที่เราทำการขอจองไว้เพื่อป้องกันการโดนย้ายหรือเขียนทับ

XmsUnlockHandle( ) ฟังก์ชันนี้ใช้ในการ unlock หน่วยความจำ EMB ที่เราล็อกไว้

XmsRelease( ) ฟังก์ชันนี้ใช้ในการปลดปล่อยหน่วยความจำ EMB ที่จองไว้โดยฟังก์ชัน XmsAllocate( )

XmsHandleStatus( ) ฟังก์ชันนี้ใช้ในการตรวจสอบ handle status อย่างเช่นว่ามีจำนวน handle เหลืออยู่เท่าไหร่ หรือมีการ lock handle จำนวนเท่าไหร่

XmsMoveDataToXms( ) ฟังก์ชันนี้ใช้ในการเคลื่อนย้ายข้อมูลจากหน่วยความจำ convention memory ไปยังหน่วยความจำ EMB

XmsGetDataFromXms( ) ฟังก์ชันนี้ใช้ในการเคลื่อนย้ายข้อมูลจากหน่วยความจำ EMB ไปยังหน่วยความจำ convention memory

XmsError( ) ฟังก์ชันนี้ใช้ในการแสดงเออเรอเมสเสจเมื่อฟังก์ชันที่เราใช้งานเกิดเออเรอขึ้น

สำหรับการประยุกต์ใช้งานฟังก์ชัน XMS นี้ผมนำตัวอย่างแบบง่าย ๆ มาแสดงให้ดูดังในรูปที่ 1 ซึ่งจะเป็นการทดสอบฟังก์ชันต่าง ๆ ทั้งหมด และอีกส่วนหนึ่งจะเป็นซอร์สโค้ดของฟังก์ชันทั้งหมดผมเขียนด้วยภาษา C+ in line assembly และทำการทดสอบกับเทอร์โบซี 2.0 กับเทอร์โบ C++ 3.0 ก็สามารถใช้งานได้ปกติ แต่เพื่อความสะดวกผมเลยทำเป็นไลบรารีไว้ใช้ ซึ่งไลบรารีนี้ผมใช้ TLIB ของเทอร์โบซี 2.0

ไลบรารีนี้เป็น Large model library (ดูรูปที่ 1)

วิธีการคอมพายล์โปรแกรมเรียก

```
C:>tcc -ml xmstest.c xmsl.lib
```

ในกรณีที่คุณไม่มี XMSL.LIB คุณจำเป็นจะต้องคอมพายล์ XMSROU.C ใหม่ ถ้าคุณใช้ TC2.0 จำเป็นจะต้องมี TASM.EXE หรือ MASM.EXE อย่างใดอย่างหนึ่ง

ลิสติงที่ 1 XMS.H

```
/*****************************************************************************
*                                                                            *
* Program name      : XMS.H                                                  *
*                                                                            *
* Version           : 00.00                                                  *
*                                                                            *
* Mm/dd/yy          : 12/16/93                                               *
*                                                                            *
* Programmer        : Mr. Visan Lauhasukpisarn                               *
*                                                                            *
*****************************************************************************/

/*============================================================================
=                              Include *.H                                   =
============================================================================*/

typedef                 unsigned int            UINT        ;
typedef                 unsigned char           UCHAR       ;
typedef     unsigned long   ULONG   ;

/*============================================================================
=                              Global Variable                               =
============================================================================*/

typedef struct
   {
   UINT   version       ;
   UINT   revision      ;
   UINT   hmaflag       ;
   } XMSINFO   ;

typedef struct
   {
      UINT   totalfree     ;
      UINT   largestblock  ;
   } XMSMEM    ;

typedef struct
   {
      UCHAR  lockcount  ;
      UCHAR  freehandle ;
   } XMSSTATUS  ;

union  XMSoff                 /* XMS mem. block offset decl.     */
         {
   ULONG  offset          ; /* XMS block offset                */
   char   *address        ; /* Conventional memory address     */
   }  ;

typedef struct                /* XMS memory block control structure */
         {
   ULONG  bsize           ; /* XMS mem. blck size (byte) ; even   */
   int    src_fd          ; /* XMS memory blck source Handle      */
   union XMSoff src_off   ; /* Src offs or conv.mem.addr. if Hnd=0 */
   int    dest_fd         ; /* XMS memory blck destination Handle */
   union XMSoff dest_off  ; /* Dest offs or conv.mem.addr if Hnd=0 */
   } XMSCOPYBLOCK     ;

/*============================================================================
=                              Function Prototype                            =
============================================================================*/

UINT   XmsAllocate( ULONG )      ;
int    XmsAccess( void )         ;
void   XmsError( UCHAR * )       ;
int    XmsGetDataFromXms( int, ULONG, ULONG, UCHAR * )   ;
int    XmsHandleStatus( UINT, XMSSTATUS * )              ;
int    XmsInfo( XMSINFO * )      ;
int    XmsLockHandle( UINT )     ;
int    XmsMemoryAvailable( XMSMEM * )    ;
int    XmsMoveDataToXms( UCHAR *, ULONG, int, ULONG )    ;
int    XmsReAllocate( UINT, ULONG )      ;
int    XmsRelease( UINT )        ;
int    XmsUnlockHandle( UINT )   ;

/*============================================================================
=                                                                            =
============================================================================*/
```

ลิสติงที่ 2 XMSROU.C

```
/*************************************************************************
*                                                                        *
* Program name      : XMSROU.C                                           *
*                                                                        *
* Version           : 00.00                                              *
*                                                                        *
* Mm/dd/yy          : 12/16/93                                           *
*                                                                        *
* Programmer        : Mr. Visan Lauhasukpisarn                           *
*                                                                        *
*************************************************************************/

/*========================================================================
=                          Include *.H                                   =
========================================================================*/

#include<dos.h>
#include  <alloc.h>
#include  <stdio.h>
#include<bios.h>
#include  <string.h>
#include  <conio.h>

#include  "xms.h"

/*========================================================================
=                          Global Variable                               =
========================================================================*/

static UCHAR  XMSErrorCode       ; /* Error code variable */
static void   (*XMSAddr)( ) = NULL  ; /* Entry point for HIMEM.SYS driver   */
static XMSCOPYBLOCK  block       ; /* Structure for moving to and from XMS */

typedef struct
  {
    UCHAR  errorcode  ;        /* XMS error code       */
    UCHAR  index      ;        /* Index to error message */
  } ErrorConv  ;

char   *XMSErrorMessage[ ] =
  {
    "(00H) No XMM error"          ,
    "(10H) Must call XMSInit( ) first",
    "(80H) Unknown function"      ,
    "(81H) Found VDISK driver"    ,
    "(82H) A20 error occurred"    ,
    "(8EH) General XMM error"     ,
    "(8FH) Fatal XMM error"       ,
    "(90H) HMA does not exist"    ,
    "(91H) HMA already in use"    ,
    "(92H) DX < /HMAMIN = parameter",
    "(93H) HMA not allocated"     ,
    "(94H) A20 enabled"           ,
    "(A0H) All XMM allocated"     ,
    "(A1H) XMM handle  not avail" ,
    "(A2H) Not a valid handle"    ,
    "(A3H) Not a valid source handle" ,
    "(A4H) Not a valid source offset" ,
    "(A5H) Not a valid dest handle" ,
    "(A6H) Not a valid dest offset" ,
    "(A7H) Not a valid size"      ,
    "(A8H) Superpose error"       ,
    "(A9H) Parity error"          ,
    "(AAH) Block not locked"      ,
    "(ABH) Block locked"          ,
    "(ACH) Locking counter overflow",
    "(ADH) Locking failed"        ,
    "(B0H) Smaller UMB availalable" ,
    "(B1H) UMB not available"     ,
    "(B2H) UMB num not valid"     ,
    "(??H) Unknown error code"    ,
  } ;
/*========================================================================
=                          Function Prototype                            =
========================================================================*/
```

→

ลิสติงที่ 2 XMSROU.C (ต่อ)

```
/*=========================================================================
=                                XmsAccess( ).                            =
=========================================================================*/

/*
  XmsAccess( ) -  Check the presence of the XMS driver

  Return  0  If success.
       -1  If error.
*/

int   XmsAccess( void )

{
UINT   l_segment = 0   ;
UINT   l_offset  = 0   ;

  asm   mov   ax, 4300h
  asm   int   2Fh
  asm   cmp   al, 80h
  asm   jne   NotOk
  asm   mov   ax, 4310h
  asm   int   2Fh
  asm   mov   l_offset, bx
  asm   mov   l_segment, es
NotOk :
  if( l_segment == 0 ) {
    return -1 ;
  }
  XMSAddr =  MK_FP( l_segment, l_offset )  ;
  return  0  ;
}

/*=========================================================================
=                                XmsAllocate( )                           =
=========================================================================*/

/*
  XmsAllocate( )    -  Allocate block of XMS memory
  Return  handle  if success.
       0     if error
*/

UINT   XmsAllocate( size )
     ULONG    size ;
{
UINT   fd          ;
UINT   allocsize     ;

  XMSErrorCode   =  0  ;

  allocsize  =  size/1024L  ;
  if( ( size%1024L ) != 0 )  {
    allocsize  =  allocsize++ ;
  }

  asm   mov   ah, 9
  asm   mov   dx, allocsize
  asm   call  XMSAddr
  asm   or    ax, ax
  asm   jnz   AllocOk
  asm   mov   XMSErrorCode, bl
  asm   xor   dx, dx

  return  0  ;

AllocOk:

  asm   mov   fd, dx
  return  fd  ;
}

/*=========================================================================
=                                XmsError( )                              =
=========================================================================*/
```

→

ซอฟต์แวร์
การใช้งานหน่วยความจำแบบ Extended ด้วยภาษาซี

ลิสติงที่ 2 XMSROU.C (ต่อ)

```
void   XmsError( str )
     UCHAR  *str  ;
{
int   i, index  ;
static ErrorConv  ErrorTable[ ]  =  {  { 0x0, 0 } , { 0x10, 1 }, { 0x80, 2 } , { 0x81, 3 } , { 0x82, 4 },
{ 0x8E, 5 }, { 0x8F, 6 }, { 0x90, 7 }, { 0x91, 8 }, { 0x92, 9 }, { 0x93, 10 }, { 0x94, 11 },
{ 0xA0,12 }, { 0xA1,13 }, { 0xA2,14 }, { 0xA3, 15}, { 0xA4, 16}, { 0xA5, 17 }, { 0xA6, 18 },
{ 0xA7,19 }, { 0xA8,20 }, { 0xA9,21 }, { 0xAA, 22}, { 0xAB, 23}, { 0xAC, 24 }, { 0xAD, 25 },
{ 0xB0,26 }, { 0xB1,27 }, { 0xB2,28 } ,{ 0xFE, 29 }, { 0xFF, 29 } } ;

  for( i = 0 ; ErrorTable[ i ].errorcode != 0xFF && i < 40 ; i++ ) {
    if( ErrorTable[ i ].errorcode == XMSErrorCode ) {
      index  =  ErrorTable[ i ].index  ;
      break  ;
    }
  }

  fprintf( stderr, "%s : %s\n", str, XMSErrorMessage[ index ] )  ;
}

/*=================================================================================
=                                  XmsGetDataFromXms( )                                =
=================================================================================*/

/*
  XmsGetDataFromXms( )  -  Move block from XMS to convention memory
                  data must have been previously allocated
                  and moved to XMS
  Return  0  if success
       -1  if error
*/

int   XmsGetDataFromXms( xmshandle, xmsoffset, movlen, dest )
     int   xmshandle  ;
     ULONG  xmsoffset  ;
     ULONG  movlen    ;
     UCHAR  *dest     ;
{
UINT   status      ;

  XMSErrorCode  =  0  ;

  block.bsize   =  movlen ;
  block.src_fd  =  xmshandle  ;
  block.src_off.offset  =  xmsoffset  ;
  block.dest_fd  =  0    ;
  block.dest_off.address =  dest     ;

  asm   mov   ah, 0Bh
  asm   mov   si, offset  block
  asm   call  XMSAddr
  asm   mov   status, ax
  asm   or    ax, ax
  asm   jnz   GetOk
  asm   mov   XMSErrorCode, bl

GetOk:
  if( status == 0 )  {
    return  -1  ;
  }
  return  0  ;
}

/*=================================================================================
=                                  XmsHandleStatus( )                                =
=================================================================================*/

int   XmsHandleStatus( handle, stat )
     UINT      handle  ;
     XMSSTATUS  *stat  ;
{
XMSSTATUS  l_stat  ;
UCHAR  lockcount, freehandle  ;

  XMSErrorCode  =  0  ;
```

→

ลิสติงที่ 2 XMSROU.C (ต่อ)

```
  asm   mov   dx, handle
  asm   mov   ah, 0Eh
  asm   call  XMSAddr
  asm   or    ax, ax
  asm   jnz   StatusOk
  asm   mov   XMSErrorCode, bl

StatusOk:
  asm   mov   lockcount, bh
  asm   mov   freehandle, bl

  if( XMSErrorCode != 0 ) {
    return -1 ;
  }
  stat->lockcount = lockcount    ;
  stat->freehandle = freehandle   ;
  return 0 ;
}

/*=================================================================
=                          XmsInfo( )                            =
=================================================================*/

/*
  XmsInfo( ) - Get XMS version number, driver revision and
          HMA existence flag
  return 0  if success
      -1  if error
*/

int   XmsInfo( info )
     XMSINFO *info    ;
{
UINT   xms, xmm, hma  ;

  XMSErrorCode  = 0 ;
```

```
  asm   mov   ah, 0
  asm   call  XMSAddr
  asm   or    ax, ax
  asm   jnz   InfoOk
  asm   mov   XMSErrorCode, bl

InfoOk:
  asm   mov   xms, ax
  asm   mov   xmm, bx
  asm   mov   hma, dx

  if( XMSErrorCode != 0 ) {
    return -1 ;
  }

  info->version  =  xms ;
  info->revision =  xmm ;
  info->hmaflag  =  hma ;

  return 0 ;
}

/*=================================================================
=                          XmsLockHandle( )                      =
=================================================================*/

/*
  XmsLockHandle( )  -  Locks a block of extended memory
  Return 0  if success
      -1  if error
*/

int   XmsLockHandle( handle )
     UINT   handle ;
{
  XMSErrorCode  =  0 ;
```

→

ลิสติงที่ 2 XMSROU.C (ต่อ)

```
    asm   mov   ah, 0Ch
    asm   mov   dx, handle
    asm   call  XMSAddr
    asm   or    ax, ax
    asm   jnz   LockOk
    asm   mov   XMSErrorCode, bl
    return -1 ;

LockOk:
    return 0 ;
}

/*=================================================================
=                    XmsMemoryAvailable( )                        =
=================================================================*/

/*
    XmsMemoryAvailable( ) - Return total XMS memory available in Kbytes

    Return not zero if success
        -1     if error
*/

int   XmsMemoryAvailable( mem )
      XMSMEM    *mem  ;
{
UINT  total, largest ;

    asm   mov   ah, 8
    asm   call  XMSAddr
    asm   or    ax, ax
    asm   jnz   MemAvail
    asm   mov   XMSErrorCode, bl

MemAvail:
    asm   mov   total, dx
    asm   mov   largest, ax

    if( XMSErrorCode != 0 ) {
      return -1 ;
    }

    mem->totalfree = total    ;
    mem->largestblock = largest ;

    return 0 ;
}


/*=================================================================
=                    XmsMoveDataToXms( )                          =
=================================================================*/

/*
    XmsMoveDataToXms( ) - Move block of data from convention memory to
                  XMS memory. Data must have been previously
                  allocated
    return 0 if success
        -1 if error
*/

int   XmsMoveDataToXms( src, movlen, xmshandle, xmsoffset )
      UCHAR  *src  ;
      ULONG  movlen ;
      int    xmshandle ;
      ULONG  xmsoffset ;
{
UINT  status      ;

    XMSErrorCode = 0 ;

    block.bsize = movlen ;
    block.src_fd = 0    ;
    block.src_off.address = src
```

→

ลิสติงที่ 2 XMSROU.C (ต่อ)

```
  block.dest_fd  =  xmshandle        ;
  block.dest_off.offset  =  xmsoffset  ;

  asm    mov    ah, 0Bh
  asm    mov    si, offset block
  asm    call   XMSAddr
  asm    mov    status, ax
  asm    or     ax, ax
  asm    jnz    MoveOk
  asm    mov    XMSErrorCode, bl

MoveOk:
  if( status == 0 )  {
     return -1  ;
  }

  return 0  ;
}

/*=================================================================
=                          XmsReAllocate( )                        =
=================================================================*/

/*
  XmsReAllocate( )    -  ReAllocate block of XMS memory
  Return  handle  if success.
        -1     if error
*/

int    XmsReAllocate( handle, size )
     UINT    handle   ;
     ULONG    size  ;
{
UINT   fd          ;
UINT   allocsize     ;

  XMSErrorCode   =  0  ;

  allocsize  =  size/1024L  ;
  if( ( size%1024L ) != 0 )  {
     allocsize  =  allocsize++ ;
  }

  asm    mov    ah, 0Fh
  asm    mov    bx, allocsize
  asm    mov    dx, handle
  asm    call   XMSAddr
  asm    or     ax, ax
  asm    jnz    ReAllocOk
  asm    mov    XMSErrorCode, bl
  asm    xor    dx, dx

  return -1  ;

ReAllocOk:
  return 0   ;
}

/*=================================================================
=                          XmsRelease( )                           =
=================================================================*/

/*
  XmsRelease( )  -  Release block of XMS memory
  Return  0  if success
        -1  if error
*/

int    XmsRelease( xmshandle )
     UINT    xmshandle  ;
{
int    status = 0    ;

  XMSErrorCode   =  0  ;
```

→

ซอฟต์แวร์
การใช้งานหน่วยความจำแบบ Extended ด้วยภาษาซี

ลิสติงที่ 2 XMSROU.C (ต่อ)

```
    asm    mov    ah, 0Ah
    asm    mov    dx, xmshandle
    asm    call   XMSAddr
    asm    or     ax, ax
    asm    jnz    ReleaseOk
    asm    mov    XMSErrorCode, bl

ReleaseOk:

    asm    mov    status, ax

    if( status == 0 )  {
        return -1 ;
    }
    return 0  ;
}

/*=================================================================
=                    XmsUnlockHandle( )                          =
=================================================================*/

/*
    XmsUnlockHandle( )  -  Unlocks a block of extended memory

    Return 0  if success
           -1  if error
*/

int    XmsUnlockHandle( handle )
       UINT   handle ;
{

    XMSErrorCode  =  0  ;

    asm    mov    ah, 0Dh
    asm    mov    dx, handle
    asm    call   XMSAddr
    asm    or     ax, ax
    asm    jnz    UnlockOk
    asm    mov    XMSErrorCode, bl
    return -1 ;

UnlockOk:

    return 0  ;
}
```

■

ลิสติงที่ 3 XMSTEST.C

```
/*****************************************************************
*                                                               *
* program name      : Test XMS memory function ....             *
*                                                               *
* version           : 00.00                                     *
*                                                               *
* mm/dd/yy          : 11/24/93                                  *
*                                                               *
* programmer        : Mr. Visan Lauhasukpisarn                  *
*                                                               *
*****************************************************************/

/*=================================================================
=                    Include *.H                                 =
=================================================================*/
```

→

ลิสติงที่ 3 XMSTEST.C (ต่อ)

```
#include   <stdio.h>

#include   "xms.h"

/*=============================================================================
=                              Global Variable                                =
=============================================================================*/

#define      BUFFERSIZE      80L

/*=============================================================================
=                              Function Prototype                             =
=============================================================================*/

/*=============================================================================
=                                   main( )                                   =
=============================================================================*/

void   main( void )

{
int   ret   ;
XMSINFO info   ;
XMSMEM  mem    ;
XMSSTATUS  stat   ;
UINT   xmshandle  ;
int    totalxms      ;
int    largestxms    ;
int    count, i      ;
char   buffer1[ BUFFERSIZE + 1 ], buffer2[ BUFFERSIZE + 1 ] ;
long   allocsize  ;

   if( XmsAccess( ) == -1 ) {
      fprintf( stderr, "XmsAccess failed\n" )    ;
      exit( 2 )  ;
   }
   printf( "\nXMS Driver installed.\n\n" )   ;

   if( XmsInfo( &info ) == -1 ) {
      XmsError( "XmsInfo" )   ;
      exit( 2 )  ;
   }
   printf( "XMS Version         : %x.%02x   installed\n"
         "XMS Driver revision  : %x.%02x   installed\n"
         "HMA existence        : %x \n\n"    ,
         info.version >> 8 , info.version & 0x00ff, info.revision >> 8, info.revision & 0x00ff , info.hmaflag ) ;

   if( ( ret = XmsMemoryAvailable( &mem ) ) == -1 ) {
      XmsError( "XmsMemoryAvailable" )  ;
      exit( 2 )  ;
   }
   printf( "Total XMS memory available       = %d  Kbytes\n" , mem.totalfree ) ;
   printf( "Largest block XMS memory available =  %d  Kbytes\n" , mem.largestblock ) ;

   count  =  2   ;
   allocsize  =  ( long )( BUFFERSIZE * count )  ;

   if( ( xmshandle = XmsAllocate( allocsize ) ) == 0 )  {
      XmsError( "XmsAllocate" ) ;
      exit( 2 )  ;
   }
   printf( "Allocate memory = %lu\n" , allocsize ) ;
   printf( "xmshandle       = %u\n" , xmshandle )      ;

   if( ( ret = XmsHandleStatus( xmshandle, &stat ) ) == -1 )  {
      XmsError( "XmsHandleStatus" ) ;
      exit( 2 )  ;
   }
   printf( "Lock count      = %d\n" , stat.lockcount )    ;
   printf( "Free handle     = %d\n" , stat.freehandle )   ;

   /*  Move data from convention memory to xms memory  */

   for( i = 0 ; i < count ; i++ )  {
      sprintf( buffer1, "Test data block count xms = %05d\n", i ) ;
```

→

ลิสติงที่ 3 XMSTEST.C (ต่อ)

```
    printf( "|Move To |%s", buffer1 )    ;

    if( ( ret = XmsMoveDataToXms( buffer1, BUFFERSIZE, xmshandle, 0L + ( long )( BUFFERSIZE * i ) ) ) == -1 )  {
      XmsError( "XmsMoveDataToXms" ) ;
      XmsRelease( xmshandle )   ;
      exit( 2 )  ;
    }
  }

  /*  Get data from xms memory to convention memory   */

  for( i = 0 ; i < count ; i++ ) {
    if( ( ret = XmsGetDataFromXms( xmshandle, 0L + ( long )( BUFFERSIZE * i ), BUFFERSIZE, buffer2 ) ) == -1 )  {
      XmsError( "XmsGetDataFromXms" ) ;
      XmsRelease( xmshandle )   ;
      exit( 2 )  ;
    }
    printf( "| Get From|%s", buffer2 )      ;
  }

  /*  Reallocate memory block   */

  if( ( ret = XmsReAllocate( xmshandle, allocsize + 2000 ) ) == -1 )  {
    XmsError( "XmsReAllocate" ) ;
    exit( 2 )  ;
  }

  if( ( ret = XmsMemoryAvailable( &mem ) ) == -1 ) {
    XmsError( "XmsMemoryAvailable" )  ;
    exit( 2 )  ;
  }
  printf( "Total XMS memory available       = %d  Kbytes\n" , mem.totalfree ) ;
  printf( "Largest block XMS memory available =  %d  Kbytes\n" , mem.largestblock ) ;

  if( ( ret = XmsLockHandle( xmshandle ) ) == -1 ) {
    XmsRelease( xmshandle )    ;
    XmsError( "XmsLockHandle" ) ;
      exit( 2 )  ;
    }

    if( ( ret = XmsHandleStatus( xmshandle, &stat ) ) == -1 )  {
      XmsError( "XmsHandleStatus" ) ;
      exit( 2 )  ;
    }
    printf( "Before Lock\n" ) ;
    printf( "Lock count     = %d\n" , stat.lockcount )    ;
    printf( "Free handle    = %d\n" , stat.freehandle )   ;

    if( ( ret = XmsUnlockHandle( xmshandle ) ) == -1 ) {
      XmsRelease( xmshandle )    ;
      XmsError( "XmsUnlockHandle" ) ;
      exit( 2 )  ;
    }

    if( ( ret = XmsHandleStatus( xmshandle, &stat ) ) == -1 )  {
      XmsError( "XmsHandleStatus" ) ;
      exit( 2 )  ;
    }
    printf( "After Unlock\n" )   ;
    printf( "Lock count     = %d\n" , stat.lockcount )    ;
    printf( "Free handle    = %d\n" , stat.freehandle )   ;

    XmsRelease( xmshandle )      ;
    exit( 0 )     ;
}
```